Министерство науки и высшего образования
Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

Пояснительная записка

К курсовому проектированию
По курсу «Логика и основы алгоритмизации в инженерных задачах»
на тему «Реализация алгоритма Дейкстры»

Выполнил :
Студент группы 22ВВС1
Костин Максим

Приняла:
к.т.н Юрова О.В.

Пенза 2023

Содержание

Реферат

Отчет 37 стр, 15 рисунков, 1 таблица.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ДЕЙКСТРЫ, РАССТОЯНИЕ.

Цель исследования – разработка программы, способной находить кратчайшие пути от одной из вершин графа до всех остальных, с помощью алгоритма Дейкстры.

В работе рассмотрен алгоритм Дейкстры. Установлено, что с помощью данного алгоритма можно найти расстояния от начальной вершины до всех остальных, а также находить кратчайшие пути до этих вершин.

Введение

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году.

Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании, например, его используют протоколы маршрутизации OSPF и IS-IS.

Среда разработки данного проекта Microsoft Visual Studio 2022, язык программирования – Си.

Целью данной курсовой работы является разработка программы, способной выполнять алгоритм Дейкстры.

Постановка задачи

Требуется разработать программу, способную выполнять алгоритм Дейкстры, то есть искать кратчайшие пути от одной вершины до всех остальных.

В качестве исходного графа будет выступать матрица смежности, которая будет создаваться тремя способами: 1)Случайной генерацией; 2)Ручным вводом данных с клавиатуры; 3)Загрузкой из файла. В первых двух случаях программа будет спрашивать у пользователя, какого типа ему нужен граф: ориентированный или неориентированный.

Необходимо предусмотреть различные исходы поиска путей, чтобы не возникало ошибок, прерывающих выполнение программы.

Устройство ввода – клавиатура.

Теоретическая часть задания

Граф G (рисунок 1) задан 5 вершинами и ребрами между этими вершинами, стрелки показывают что, граф ориентирован, и указывают на направление путей, а числа над ребрами указывают на вес рёбер.

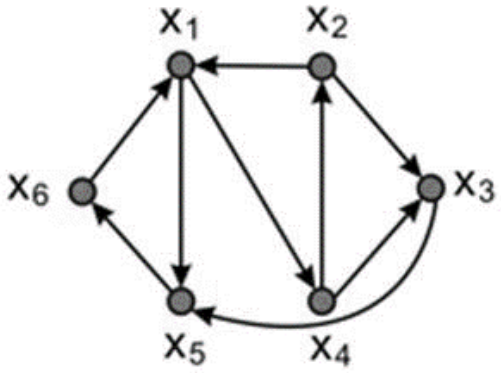


Рисунок 1 – ориентированный граф

При представлении этого графа матрицей смежности информация о ребрах графа хранится в матрице n*n (n – количество вершин графа), где присутствие пути обозначается весом ребра между вершинами.

Алгоритм Дейкстры — это алгоритм, который вычисляет кратчайший путь от одной исходной вершины ко всем остальным вершинам.

Данный алгоритм применим, к графам с взвешенными рёбрами. Он не допускает рёбра с отрицательным весом. Длиной пути является сумма весов рёбер, входящих в этот путь.

Основная идея алгоритма заключается в том, что на каждом шаге помечается определенным образом выбранная вершина, а далее просматриваются все последующие (ещё не отмеченные) вершины графа и вычисляется длина пути до каждой из них от начальной точки.

Описание алгоритма программы

Для реализации данного алгоритма понадобились 3 динамических массива. Первый int* d – для записи минимальных расстояний от начальной вершины для все остальных , расстояние до самой начальной вершины помечается, как 0. Второй int* v – нужен для того чтобы знать посещена вершина или нет 0 посещена, 1 – нет. Третий int* ver – для помещения в него вершин кратчайшего пути.

В саму функцию void dejkctr(int** matr, int n) передается матрица смежности(int** matr) и количество вершин в графе (int n). В качестве начальной вершины пользователю предлагают выбрать одну из возможных, если же выбрана не существующая вершина, то программа просит заново указать вершину, только уже существующую. Дальше идет инициализация массива непосещенных вершин и массива расстояний до них. И после сразу заполняется расстояние до начальной вершины, оно равно 0.

Дальше начинается алгоритм поиска расстояний. Сначала инициализируются две переменные min и minindex в них записывается максимально возможное целое число. Дальше начинается цикл, в котором, если вершина не была посещена и вес меньше чем в min, то мы переприсваиваем значения min и minindex. После добавляется найденный минимальный вес к текущему весу вершины и сравнивается с текущим минимальным весом вершины. Так алгоритм работает, пока не заполнится кратчайшими расстояниями массив d. И следом массив выводится на экран.

Когда все вышеперечисленное будет сделано, программа спросит хотим ли мы просмотреть кратчайшие пути от начальной вершины до указанной. Если да, то мы вводим вершину, до которой хотим просмотреть путь. В массив ver в самое начало помещается вершина, до которой нам нужен кратчайший путь. После в переменную weight записывается расстояние от начальной вершины до той, до которой мы ищем кратчайший путь. Дальше запускается цикл while, пока переменная конечной вершины не

станет начальной. В цикле просматриваются все вершины, если есть связь, то определяем вес пути из предыдущей вершины, если вес совпал с рассчитанным, то значит из этой вершины и был переход, следом сохраняем новый вес, сохраняем предыдущую вершину и записываем ее в массив ver. Если связи нет, увеличиваем считчик переменной inff и если этот счет будет >= количеству вершин, то расстояния до указанной вершины нет.

Дальше программа еще раз спросит о желании посмотреть пути до вершин, если пользователь не хочет, то функция завершает свою работу.

Ниже представлен псевдокод функции void dejkctr(int** matr, int n):

```
1. int* v = (int*)malloc(sizeof(int) * n)
2. int temp, minindex, min
3. int begin = 0

   4.int* d = (int*)malloc(sizeof(int) * n)
   5.Начало цикла
            6. Ввод начальной вершины
            7. если (begin < 1 || begin > n)
                  8.Вывести «несуществующая вершина»
    9.пока (begin < 1 || begin > n);
    10.begin -= 1
    11. для i = 0 пока i < n выполнять i++{
          12.d[i] = INT_MAX;
          13.v[i] = 1
}
    14. d[begin] = 0
       15. Начало цикла {
          16.minindex = INT_MAX
          17.min = INT_MAX
          18.для int i = 0 пока i < n выполнять i++
             19.Если вершину ещё не обошли и вес меньше min
                  20.То{
                        21. min = d[i]
                        22. minindex = i;
           }
```

23.Если переменная minindex != INT_MAX
24.то
25для int i = 0 пока i < n выполнять i++
{
26. Если matr[minindex][i] > 0
{
27.temp = min + matr[minindex][i];
28.Если текущий минимальный вес меньше веса в d[i]
29.то
30.d[i] = temp;

}
}
31.v[minindex] = 0;

32.пока minindex < INT_MAX выполнять

33. Вывести «кратчайшие расстояние до вершин»
34.для int i = 0 пока i < n выполнять i++) {
35.Если расстояние = ∞
36.то
37.Вывести «нет пути в вершину»
38. иначе
39.Вывести «расстояния до вершин»
}

40.int vibor = 0;
41.int inff = 0;
42.Начало цикла
43.Вывести «Хотим ли посмотреть кратчайшие пути?»
44. Ввод
45.Если vibor > 1 || vibor < 0
46. ТО
47. Вывести «Ошибка»
48.Выполнять пока vibor > 1 || vibor < 0
49. если vibor == 1
50.То

51.Начало цикла

52.int end = 0;

53. Начало цикла

54.Ввод номера вершины до которой найти путь

55. Если end > n || end < 1

56.То

57. Вывести «Ошибка»

58.Выполнять пока (end > n || end < 1)

59.Если расстояние, до которого ищем путь = ∞

60.То

61.Вывести «нет пути!»

62.Иначе

63.int* ver = (int*)malloc(sizeof(int) * n);

64.end -= 1;

65.ver[0] = end + 1

66.int k = 1

67. вес конечной вершины int weight = d[end]

68.Выполнять пока не дошли до начальной вершины

69.для int i = 0 пока i < n выполнять i++)

70.если связь есть

71. То{

72.int temp = weight - matr[i][end]

73.если вес совпал с рассчитанным

74.То{

75.weight = temp

76.end = i

77.ver[k] = i + 1

78.k++;

}

}

}

79.Для int i = k – 1 пока i >= 0 выполнять i--

80.Вывод пути

}

81. Освобождение памяти массива ver

82.Начало цикла
83.Ввод «хотите ли продолжить?»
84.Если vibor > 1 || vibor < 0
85.То
86.Вывод «Ошибка»
87.Выполнять пока vibor > 1 || vibor < 0
88.Конец цикла просмотра путей

89.Освобождение памяти массива v
90.Освобождение памяти массива d

Полный код программы в приложении А.

Описание программы

Для написания данной программы был использован язык программирования Си.

Данная программа является многомодульной , поскольку состоит из нескольких функций: dejkctr, graph_1, graph_2, graph_3, save_graph, main.

Работа программы начинается с выбора варианта работы с графом (рисунок 1). Есть три варианта: случайно сгенерировать граф, создать самому вручную и загрузить граф из файла.

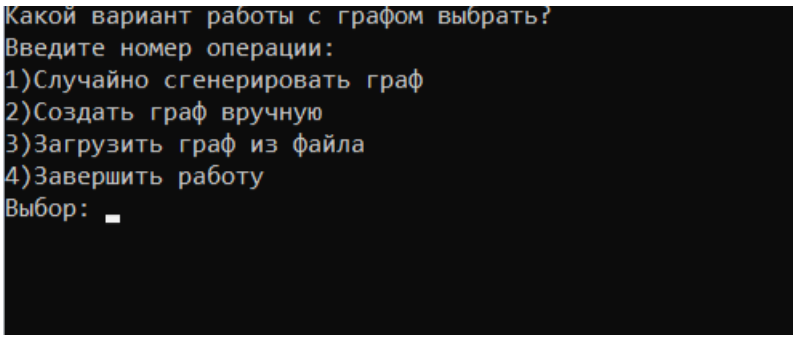


Рисунок 2 – меню программы

Рассмотрим для начала вариант случайной генерации. Сначала программа предлагает выбрать количество вершин графа, после указать предел значений весов в графе, следом нужно указать, какой граф нужно сгенерировать ориентированный или неориентированный.

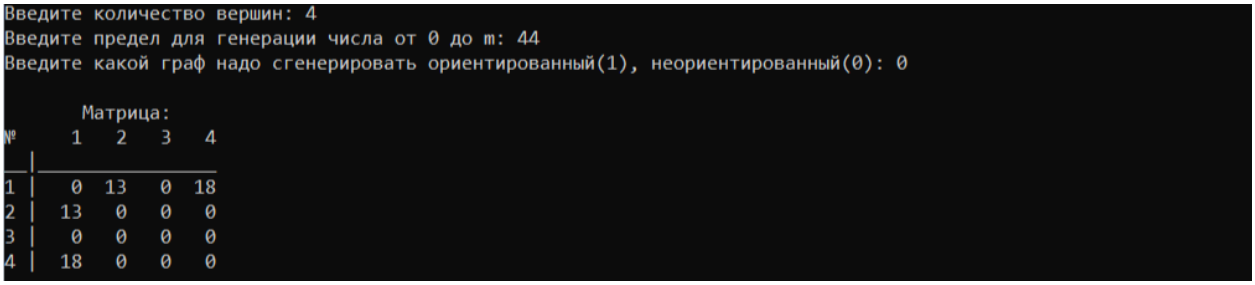


Рисунок 3 – случайная генерация графа

После генерации матрицы смежности графа, программа предлагает выбрать начальную вершину, с которой будет выполняться алгоритм

Дейкстры. После выбора вершины происходит работа алгоритма и выводятся кратчайшие расстояния до других вершин.

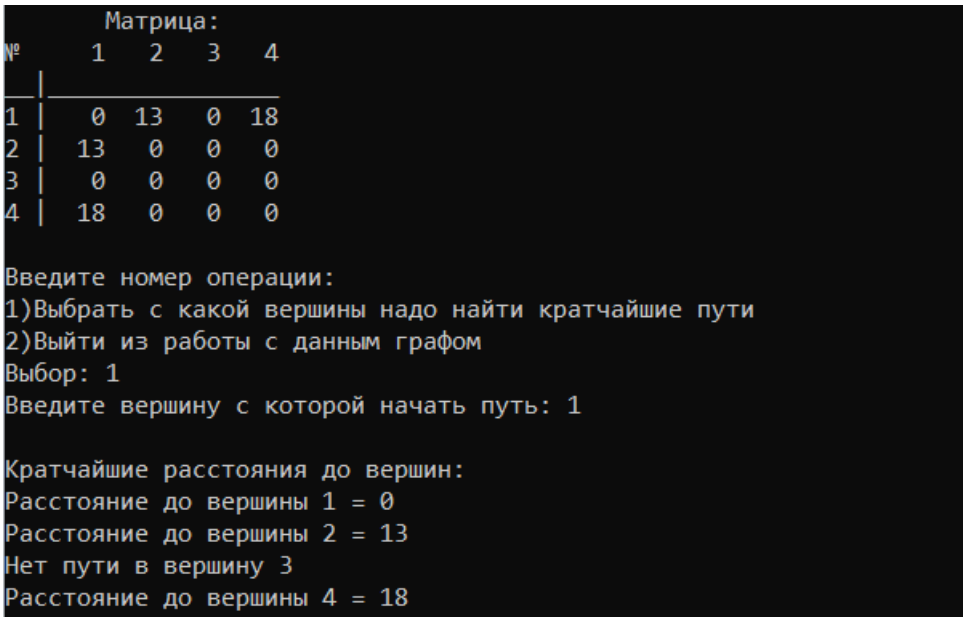


Рисунок 4 – вывод кратчайших расстояний

Дальше программа предлагает просмотреть кратчайшие пути до вершин, если пользователь согласен, то надо указать до какой вершины искать путь. После того как путь будет выведен, можно просмотреть и другие пути, либо выйти из алгоритма.

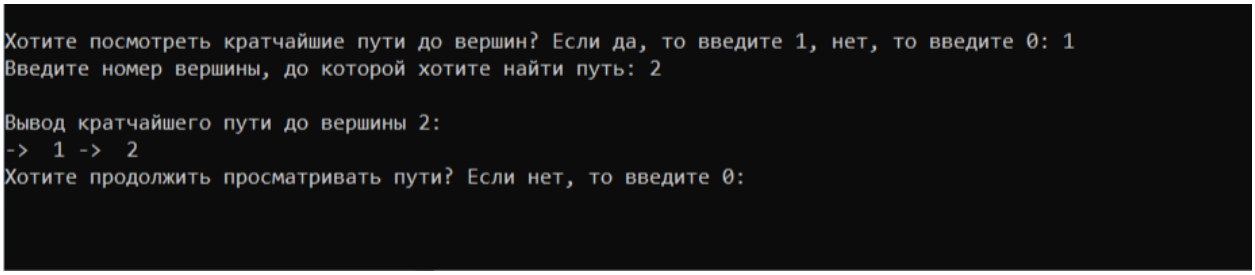


Рисунок 5 – просмотр кратчайших путей.

Когда пользователь захочет прекратить работу с данным графом, программа предложит сохранить данную матрицу смежности в файл.

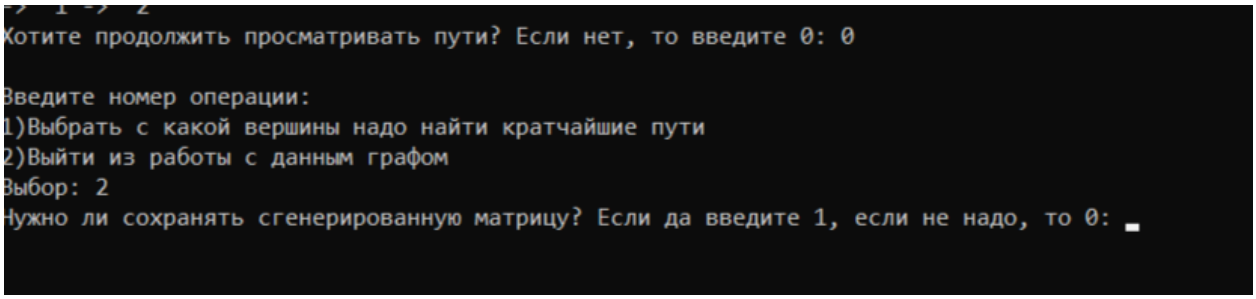


Рисунок 6 – сохранение матрицы в файл

Код случайной генерации графа:

```
srand(time(NULL));
	int n = 0;
	printf("Введите количество вершин: ");
	scanf_s("%d", &n);
	int** matr = (int**)malloc(sizeof(int*) * n);
	for (int i = 0; i < n; i++)
		matr[i] = (int*)malloc(sizeof(int) * n);

	int m = 0;
	printf("Введите предел для генерации числа от 0 до m: ");
	scanf_s("%d", &m);
	int orgr = 0;
	do {
		printf("Введите какой граф надо сгенерировать ориентированный(1), неориентированный(0): ");
		scanf("%d", &orgr);
		if (orgr > 1 || orgr < 0)
			printf("Ошибка! Введите один из предложенных вариантов!\n");
	} while (orgr > 1 || orgr < 0);
			if (orgr == 0) {
				for (int i = 0; i < n; i++) {
					for (int j = i; j < n; j++) {
						matr[i][j] = rand() % 2;
						matr[j][i] = matr[i][j];
						if (i == j)
							matr[i][j] = 0;
					}
				}
			}

			if (orgr == 1) {
				for (int i = 0; i < n; i++) {
					for (int j = i; j < n; j++) {
						matr[i][j] = rand() % 2;
						matr[j][i] = matr[i][j];
						if (i == j)
							matr[i][j] = 0;
					}
```

```
            }
        }

        if (orgr == 0) {
            for (int i = 0; i < n; i++) {
                for (int j = i; j < n; j++) {
                    if (matr[i][j] == 1) {
                        matr[i][j] = rand() % m;
                        matr[j][i] = matr[i][j];
                    }
                }
            }
        }
        int ori = 0;

        if (orgr == 1) {
            for (int i = 0; i < n; i++) {
                for (int j = i; j < n; j++) {

                    if (matr[i][j] == 1) {
                        matr[i][j] = rand() % m;
                        ori = rand() % 3;
                        if (ori == 1)
                            matr[j][i] = matr[i][j];
                        if (ori == 0)
                            matr[j][i] = 0;
                        if (ori == 2) {
                            matr[j][i] = matr[i][j];
                            matr[i][j] = 0;
                        }
                    }
                }
            }
        }
```

Дальше рассмотрим вариант создания графа вручную. Также как и в прошлом случае, пользователю предлагается ввести количество вершин графа и указать ориентированность. После этого, если граф ориентированный, то нам предлагает выбрать расстояние между вершинами и указать в какую сторону провести путь, например, только из вершины 1 в 2, только из вершины 2 в 1 или путь в обе стороны.

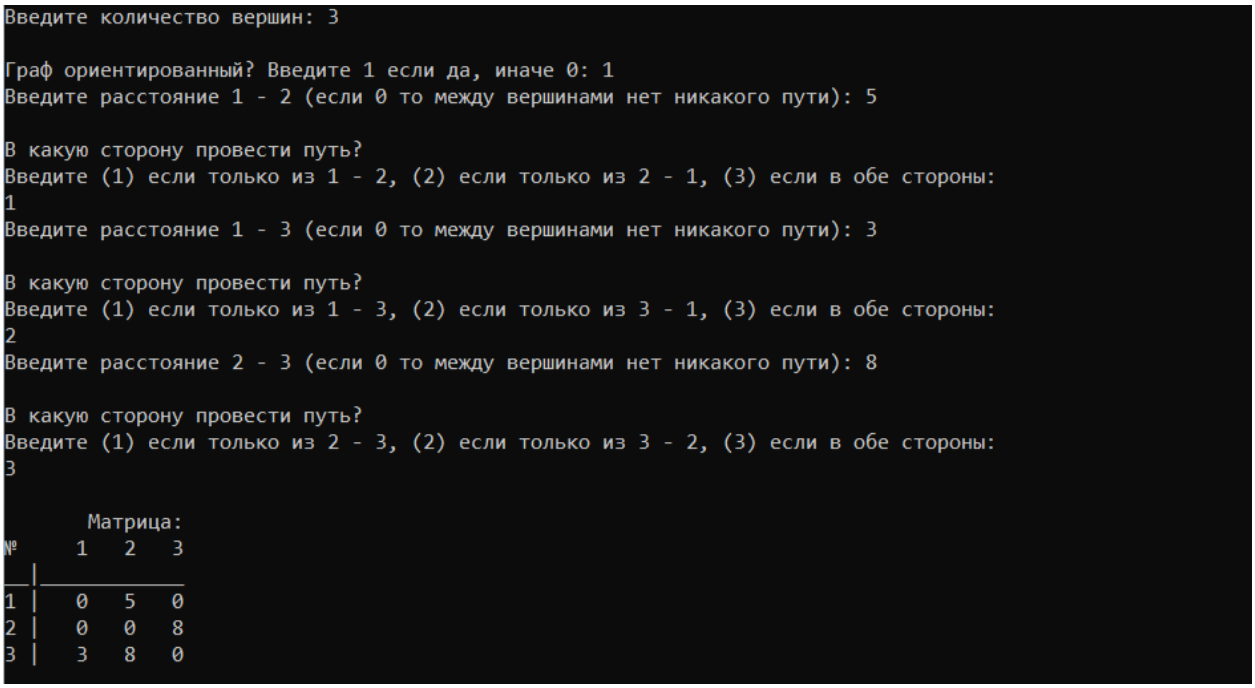


Рисунок 7 – ввод графа вручную

Дальнейшая работа с этим графом аналогична работе со случайно сгенерированным графом.

Код ручного ввода графа:

```
printf("Введите количество вершин: ");
	scanf_s("%d", &n);
	int** matr = (int**)malloc(sizeof(int*) * n);
	for (int i = 0; i < n; i++)
		matr[i] = (int*)malloc(sizeof(int) * n);

	int rebra = 0, orientier = 0;
	do {
		printf("\nГраф ориентированный? Введите 1 если да, иначе 0: ");
		scanf("%d", &orientier);
		if (orientier > 1 || orientier < 0)
			printf("Ошибка! Введите один из предложенных вариантов!\n");
	} while (orientier > 1 || orientier < 0);
	if (orientier == 0) {
		for (int i = 0; i < n; i++)
		{
			matr[i][i] = 0;
			for (int j = i + 1; j < n; j++) {
				do {
					printf("Введите расстояние %d - %d (если 0 то между вершинами нет
никакого пути): ", i + 1, j + 1);
					scanf("%d", &rebra);
					if (rebra < 0)
						printf("Ошибка! Расстояние должно быть положительным
значением!\n");
```

```
                        } while (rebra < 0);
                        matr[i][j] = rebra;
                        matr[j][i] = rebra;
                    }
                }
            }
            else {
                for (int i = 0; i < n; i++)
                {
                    matr[i][i] = 0;
                    for (int j = i + 1; j < n; j++) {
                        do {
                            printf("Введите расстояние %d - %d (если 0 то между вершинами нет никакого пути): ", i + 1, j + 1);
                            scanf("%d", &rebra);
                            if (rebra < 0)
                                printf("Ошибка! Расстояние должно быть положительным значением!\n");
                        } while (rebra < 0);

                        if (rebra != 0) {
                            do {
                                printf("\nВ какую сторону провести путь?\nВведите (1) если только из %d - %d, (2) если только из %d - %d, (3) если в обе стороны: \n", i + 1, j + 1, j + 1, i + 1);
                                scanf("%d", &orientier);
                                if (orientier > 3 || orientier < 1)
                                    printf("Ошибка! Введите один из предложенных вариантов!\n");
                            } while (orientier > 3 || orientier < 1);
                            if (orientier == 1) {
                                matr[i][j] = rebra;
                                matr[j][i] = 0;
                            }
                            if (orientier == 2) {
                                matr[i][j] = 0;
                                matr[j][i] = rebra;
                            }
                            if (orientier == 3) {
                                matr[i][j] = rebra;
                                matr[j][i] = rebra;
                            }
                        }
                        else {
                            matr[i][j] = rebra;
                            matr[j][i] = rebra;
                        }
                    }
                }
            }
```

Третий вариант работы с графом это загрузка графа из файла.

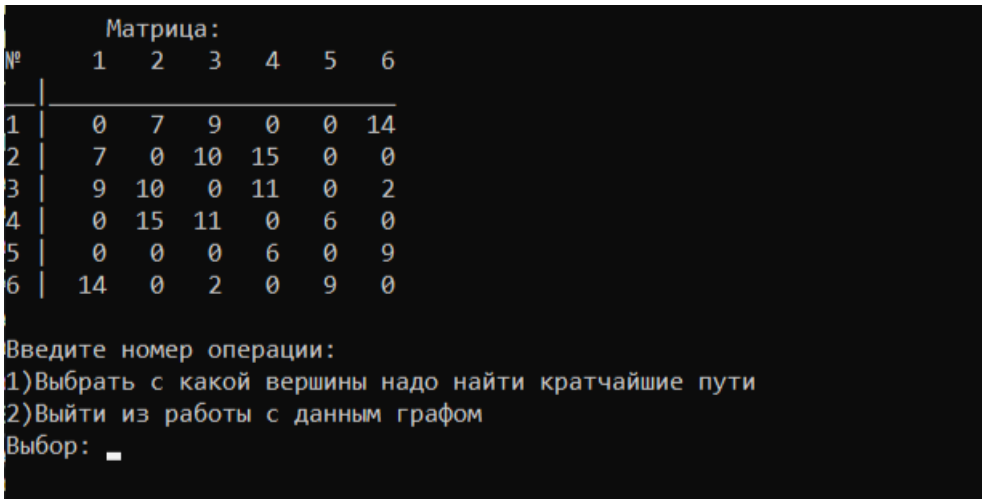


Рисунок 8 – загруженная из файла матрица

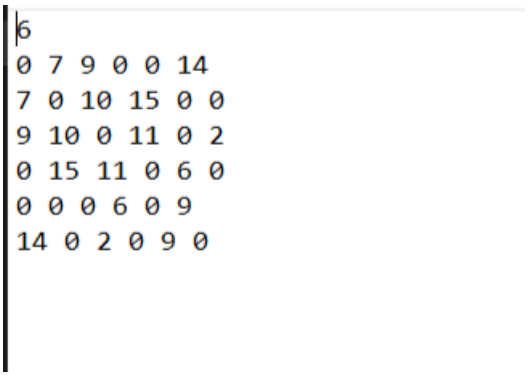


Рисунок 9 – файл с матрицей смежности

Код загрузки графа из файла:

```
FILE* F, *S;
        int n = 0;
        int vibor = 0;
        F = fopen("Matrica.txt", "r");
        if (F == NULL) {
                perror("Ошибка при открытии файла");
                while (1) {
                        printf("Хотите создать файл? Введите 1 если да, если нет, то 0:");
                        scanf("%d", &vibor);
                        switch (vibor) {
                        case 1: {
                                        S = fopen("Matrica.txt", "w");
                                        printf("Файл создан, теперь вручную заполните матрицу, сначала
укажите количество вершин, а потом и саму матрицу.");
                                        _getch();
                                        fclose(S);
                                        return 0;
                                }
                        case 0:
                                return 0;
                        default:
                                printf("Ошибка! Не существующий номер операции!Повторите попытку.\n");
```

```
                }
            }
        }

        fscanf(F, "%d", &n);
        if (n == 0) {
            printf("Заполните матрицу! Для этого в  файле 'Matrica.txt' введите сначала количество вершин, а потом и саму матрицу.");
            _getch();
            return 0;
        }
        int** matr = (int**)malloc(sizeof(int*) * n);
        for (int i = 0; i < n; i++)
            matr[i] = (int*)malloc(sizeof(int) * n);

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                matr[i][j] = 0;
        }


        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                fscanf(F, "%d", &matr[i][j]);
                if (matr[i][j] < 0) {
                    printf("Ошибка! Данный алгоритм работает только с ребрами положительного веса!\nИзмените значения на положительные!");
                    _getch();
                    return 0;
                }
            }
        }

        fclose(F);
```

И, наконец, можно просто завершить работу и выйти из программы.

Тестирование

Среда разработки Microsoft Visual Studio 2022 предоставляет все

средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

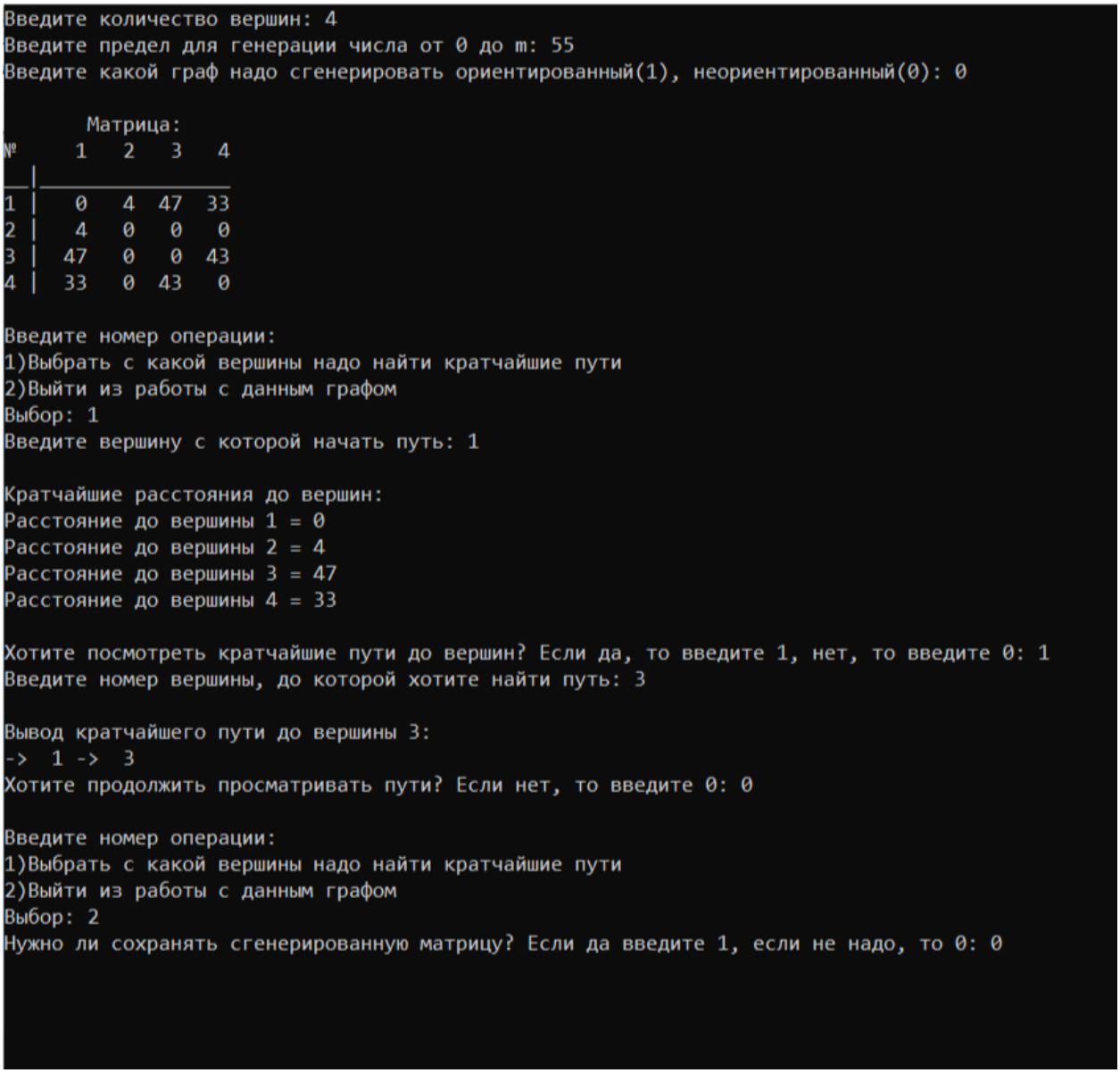


Рисунок 10 – тестирование работы с графом при случайной генерации

```
Введите количество вершин: 4

Граф ориентированный? Введите 1 если да, иначе 0: 0
Введите расстояние 1 - 2 (если 0 то между вершинами нет никакого пути): 5
Введите расстояние 1 - 3 (если 0 то между вершинами нет никакого пути): 0
Введите расстояние 1 - 4 (если 0 то между вершинами нет никакого пути): 43
Введите расстояние 2 - 3 (если 0 то между вершинами нет никакого пути): 4
Введите расстояние 2 - 4 (если 0 то между вершинами нет никакого пути): 0
Введите расстояние 3 - 4 (если 0 то между вершинами нет никакого пути): 12

       Матрица:
№     1   2   3   4
__|_______________
1 |   0   5   0  43
2 |   5   0   4   0
3 |   0   4   0  12
4 |  43   0  12   0

Введите номер операции:
1)Выбрать с какой вершины надо найти кратчайшие пути
2)Выйти из работы с данным графом
Выбор: 1
Введите вершину с которой начать путь: 1

Кратчайшие расстояния до вершин:
Расстояние до вершины 1 = 0
Расстояние до вершины 2 = 5
Расстояние до вершины 3 = 9
Расстояние до вершины 4 = 21

Хотите посмотреть кратчайшие пути до вершин? Если да, то введите 1, нет, то введите 0: 1
Введите номер вершины, до которой хотите найти путь: 2

Вывод кратчайшего пути до вершины 2:
->  1 ->  2
Хотите продолжить просматривать пути? Если нет, то введите 0: 1
Введите номер вершины, до которой хотите найти путь: 3

Вывод кратчайшего пути до вершины 3:
->  1 ->  2 ->  3
Хотите продолжить просматривать пути? Если нет, то введите 0: 0

Введите номер операции:
1)Выбрать с какой вершины надо найти кратчайшие пути
2)Выйти из работы с данным графом
Выбор:
```

Рисунок 11 – тестирование работы с графом при самостоятельном вводе

```
6
0 7 9 0 0 14
7 0 10 15 0 0
9 10 0 11 0 2
0 15 11 0 6 0
0 0 0 6 0 9
14 0 2 0 9 0
```

Рисунок 12 – матрица смежности в файле

Рисунок 13 – тестирование при загрузке матрицы из файла

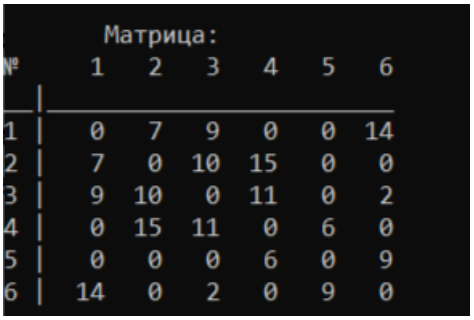
Ниже приведена таблица, которая описывает поведение программы при тестировании.

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод сообщения о выборе: 1. Случайная генерация графа 2. Ручной ввод 3. Загрузка из файла	Верно
Случайная генерация матрицы	Вывод матрицы, с указанным количеством вершин	Верно
Ручной ввод матрицы	Вывод меню с выбором графа. Вывод матрицы с введёнными данными	Верно
Считывание матрицы из файла	Вывод матрицы с данными из файла	Верно
Сохранение результата в файл	Наличие в файле результата	Верно

Таблица 1 – Описание поведения программы при тестировании

Ручной расчет задачи

Проведем проверку программы при просчете вычислений вручную.



Матрица:

№	1	2	3	4	5	6
1	0	7	9	0	0	14
2	7	0	10	15	0	0
3	9	10	0	11	0	2
4	0	15	11	0	6	0
5	0	0	0	6	0	9
6	14	0	2	0	9	0

Рисунок 14 – матрица смежности графа

Начальную вершину выберем 1. Метка самой вершины 1 полагается равной 0, метки остальных вершин – недостижимо большое число (в идеале — бесконечность). Это отражает то, что расстояния от вершины 1 до других вершин пока неизвестны. Все вершины графа помечаются как непосещенные.

Минимальную метку имеет вершина 1. Её соседями являются вершины 2, 3 и 6. Обходим соседей вершины по очереди. Первый сосед вершины 1 – вершина 2, потому что длина пути до неё минимальна. Длина пути в неё через вершину 1 равна сумме кратчайшего расстояния до вершины 1 (значению её метки) и длины ребра, идущего из 1-й во 2-ю, то есть 0 + 7 = 7. Это меньше текущей метки вершины 2 (∞), поэтому новая метка 2-й вершины равна 7. Аналогично находим длины пути для всех других соседей (вершины 3 и 6).

Все соседи вершины 1 проверены. Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит. Вершина 1 отмечается как посещенная.

Дальше алгоритм повторяется. Снова находим «ближайшую» из непосещенных вершин. Это вершина 2 с меткой 7.

Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю вершину. Соседями вершины 2 являются вершины 1, 3 и 4.

Вершина 1 уже посещена. Следующий сосед вершины 2 — вершина 3, так как имеет минимальную метку из вершин, отмеченных как не посещённые. Если идти в неё через 2, то длина такого пути будет равна 17 (7 + 10 = 17). Но текущая метка третьей вершины равна 9, а 9 < 17, поэтому метка не меняется.

Ещё один сосед вершины 2 — вершина 4. Если идти в неё через 2-ю, то длина такого пути будет равна 22 (7 + 15 = 22). Поскольку 22<10000, устанавливаем метку вершины 4 равной 22. Все соседи вершины 2 просмотрены, помечаем её как посещенную.

Повторяем шаг алгоритма, выбрав вершину 3. После её «обработки» получим следующие результаты кратчайшие расстояние от 1: до 3 вершины – 9, до 4 – 20, до 6 – 11. 3 помечаем, как посещенную. После смотри ближайшую к 3 непосещенную вершину – это 6. От 1 до 5 расстояние 11 + 9(расстояние между 6 и 5) = 20. Помечаем 6, как посещенную.

Дальше идет 4 вершина через нее расстояние до будет 26 , так как 20 < 26, то кратчайшее расстояние не заменяется.

Таким образом, кратчайшим путем из вершины 1 в вершину 5 будет путь через вершины 1 — 3 — 6 — 5, поскольку таким путем мы набираем минимальный вес, равный 20.

Таким образом, можно сделать вывод, что программа работает верно.

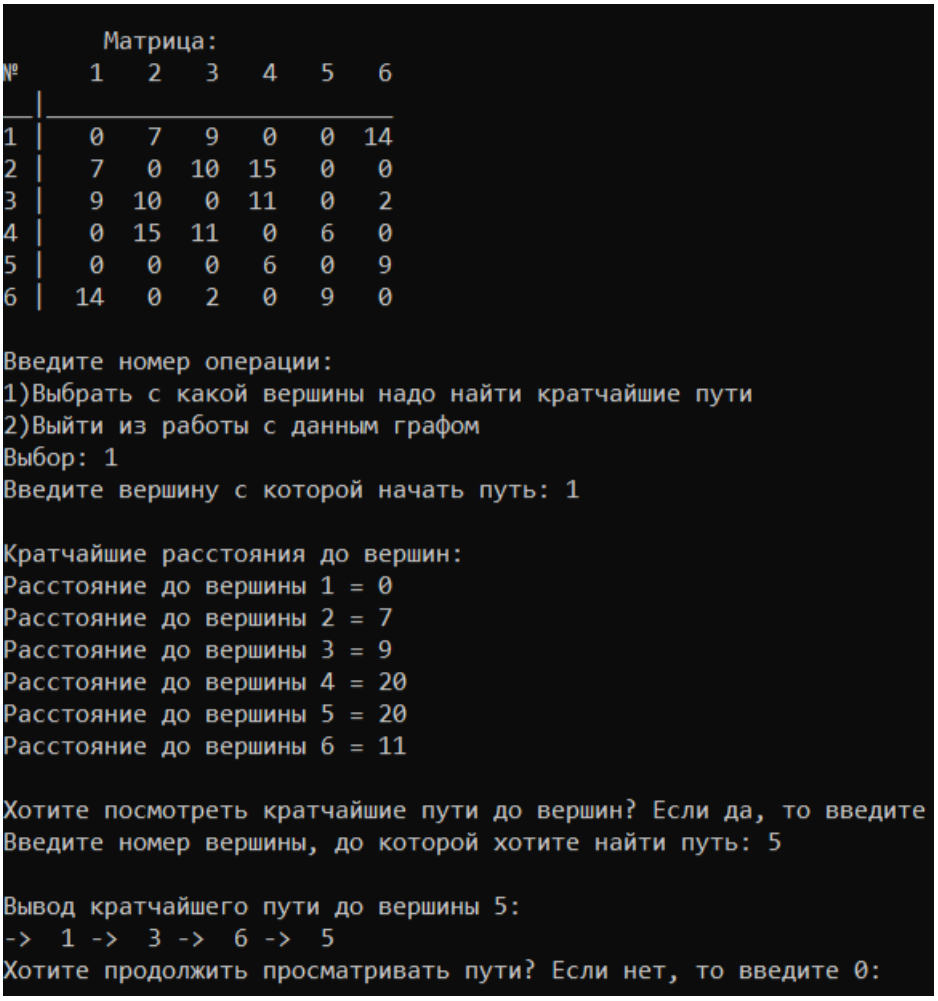


Рисунок 15 – проверка

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Дейкстры для поиска кратчайших путей от указанной вершины до всех остальных вершин.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Приобретены навыки по осуществлению алгоритма Дейкстры. Углублены знания языка программирования Си.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

Список литературы

1. Дейкстра, Э. (1959). О коротком пути в сети с взвешенными ребрами. Вклад в теорию алгоритмов (pp. 143-150).
2. Кениг, Д. (2002). Алгоритм Дейкстры. Энциклопедия алгоритмов, 1-5.
3. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Глава 22: Кратчайшие пути. Вводная в алгоритмы (3-е издание, стр. 507–534). MIT Press.
4. Седжвик, Р. (2011). Глава 16: Алгоритм Дейкстры и алгоритм Беллмана-Форда. Фундаментальные алгоритмы (2-е издание).

Приложение А.
Листинг программы.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <conio.h>
#include <time.h>
#include <windows.h>


void zast() {
	printf("\n\n\t\t\t\t---------------------------------------------------------- \n ");
	printf("\t\t\t\t| Курсовая работа студента группы 22ВВС1 Костина Максима | \n ");
	printf("\t\t\t\t---------------------------------------------------------- \n ");
	printf("\t\t\t\t|                  Алгоритм Дейкстры                  | \n ");
	printf("\t\t\t\t---------------------------------------------------------- \n ");
	Sleep(1000);
	system("cls");
}

void dejkctr(int** matr, int n) {

	int* v = (int*)malloc(sizeof(int) * n);
   int temp, minindex, min;
   int begin = 0;

   int* d = (int*)malloc(sizeof(int) * n);
	do {
		printf("Введите вершину с которой начать путь: ");
		scanf("%d", &begin);
		if (begin < 1 || begin > n)
			printf("Ошибка несуществующая вершина!\n");
	} while (begin < 1 || begin > n);
	begin -= 1;


	 for (int i = 0; i < n; i++)
	 {
	    d[i] = INT_MAX;
	    v[i] = 1;
	 }
	 d[begin] = 0;

	 do {
	    minindex = INT_MAX;
	    min = INT_MAX;
	    for (int i = 0; i < n; i++)
	    {
	       if ((v[i] == 1) && (d[i] < min))
```

```
            {
                min = d[i];
                minindex = i;
            }
        }

        if (minindex != INT_MAX)
        {
            for (int i = 0; i < n; i++)
            {
                if (matr[minindex][i] > 0)
                {
                    temp = min + matr[minindex][i];
                    if (temp < d[i])
                    {
                        d[i] = temp;
                    }
                }
            }
            v[minindex] = 0;
        }
    } while (minindex < INT_MAX);

    printf("\nКратчайшие расстояния до вершин: \n");
    for (int i = 0; i < n; i++) {
            if (d[i] == INT_MAX)
                    printf("Нет пути в вершину %d\n", i + 1);
            else
                    printf("Расстояние до вершины %d = %d\n", i + 1, d[i]);
    }

    int vibor = 0;

    do {
            printf("\nХотите посмотреть кратчайшие пути до вершин? Если да, то введите 1, нет, то введите
0: ");
            scanf("%d", &vibor);
            if (vibor > 1 || vibor < 0)
                    printf("Ошибка! Введите один из предложенных вариантов!\n");
    } while (vibor > 1 || vibor < 0);
    if (vibor == 1) {
            do {
                    int end = 0;
                    do {
                            printf("Введите номер вершины, до которой хотите найти путь: ");
                            scanf("%d", &end);
                            if (end > n || end < 1)
                                    printf("Ошибка несуществующая вершина!\n");
                    } while (end > n || end < 1);

                    if (d[end - 1] == INT_MAX) {
```

```
                printf("Нет пути!");
            }
            else {
                printf("\nВывод кратчайшего пути до вершины %d:\n", end);
                int* ver = (int*)malloc(sizeof(int) * n);
                end -= 1;
                ver[0] = end + 1;
                int k = 1;
                int weight = d[end];

                while (end != begin)
                {
                    for (int i = 0; i < n; i++)
                        if (matr[i][end] != 0)
                        {
                            int temp = weight - matr[i][end];
                            if (temp == d[i])
                            {
                                weight = temp;
                                end = i;
                                ver[k] = i + 1;
                                k++;
                            }
                        }
                }
                for (int i = k - 1; i >= 0; i--)
                    printf("->%3d ", ver[i]);
                free(ver);
            }
            do{
            printf("\nХотите продолжить просматривать пути? Если нет, то введите 0: ");
            scanf("%d", &vibor);
            if (vibor > 1 || vibor < 0)
                printf("Ошибка! Введите один из предложенных вариантов!\n");
        } while (vibor > 1 || vibor < 0);
        } while (vibor);
    }
    free(v);
    free(d);

}

int save_graph(int** matr, int n) {
    FILE* F;
    F = fopen("Matrica.txt", "w");
    if (F == NULL) {
        perror("Ошибка");
        return 0;
    }
```

```
	fprintf(F, "%d\n", n);
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++)
			fprintf(F, "%d ", matr[i][j]);
		fprintf(F, "\n");
	}


	fclose(F);

	return 0;
}

int graph_1() {
	srand(time(NULL));
	int n = 0;
	printf("Введите количество вершин: ");
	scanf_s("%d", &n);
	int** matr = (int**)malloc(sizeof(int*) * n);
	for (int i = 0; i < n; i++)
		matr[i] = (int*)malloc(sizeof(int) * n);

	int m = 0;
	printf("Введите предел для генерации числа от 0 до m: ");
	scanf_s("%d", &m);
	int orgr = 0;
	do {
		printf("Введите какой граф надо сгенерировать ориентированный(1), неориентированный(0): ");
		scanf("%d", &orgr);
		if (orgr > 1 || orgr < 0)
			printf("Ошибка! Введите один из предложенных вариантов!\n");
	} while (orgr > 1 || orgr < 0);
			if (orgr == 0) {
				for (int i = 0; i < n; i++) {
					for (int j = i; j < n; j++) {
						matr[i][j] = rand() % 2;
						matr[j][i] = matr[i][j];
						if (i == j)
							matr[i][j] = 0;
					}
				}
			}

			if (orgr == 1) {
				for (int i = 0; i < n; i++) {
					for (int j = i; j < n; j++) {
						matr[i][j] = rand() % 2;
						matr[j][i] = matr[i][j];
						if (i == j)
							matr[i][j] = 0;
```

```
                        }

                }
        }

        if (orgr == 0) {
                for (int i = 0; i < n; i++) {
                        for (int j = i; j < n; j++) {
                                if (matr[i][j] == 1) {
                                        matr[i][j] = rand() % m;
                                        matr[j][i] = matr[i][j];
                                }
                        }
                }
        }
        int ori = 0;

        if (orgr == 1) {
                for (int i = 0; i < n; i++) {
                        for (int j = i; j < n; j++) {

                                if (matr[i][j] == 1) {
                                        matr[i][j] = rand() % m;
                                        ori = rand() % 3;
                                        if (ori == 1)
                                                matr[j][i] = matr[i][j];
                                        if (ori == 0)
                                                matr[j][i] = 0;
                                        if (ori == 2) {
                                                matr[j][i] = matr[i][j];
                                                matr[i][j] = 0;
                                        }
                                }
                        }
                }
        }


printf("\n       Матрица:\n№  ");
for (int i = 0; i < n; i++) {
        printf(" %3d", i + 1);
}
printf("\n__|");
for (int i = 0; i < n; i++) {
        printf("____");
}
printf("\n");
for (int i = 0; i < n; ++i) {
        {
                printf("%d | ", i + 1);
```

```
                for (int j = 0; j < n; ++j)
                        printf("%3d ", matr[i][j]);
                printf("\n");
        }
    }
    int opr = 0;
    while (1) {
        printf("\nВведите номер операции:\n1)Выбрать с какой вершины надо найти кратчайшие пути \n2)Выйти из работы с данным графом\n");
        printf("Выбор: ");
        scanf_s("%d", &opr);
        switch (opr) {
        case 1:
            dejkctr(matr, n);
            break;
        case 2:
            do {
                printf("Нужно ли сохранять сгенерированную матрицу? Если да введите 1, если не надо, то 0: ");
                scanf("%d", &opr);
                if (opr > 1 || opr < 0)
                    printf("Ошибка! Введите один из предложенных вариантов!\n");
            } while (opr > 1 || opr < 0);
            if (opr == 1)
                save_graph(matr, n);
            for (int i = 0; i < n; i++) {
                free(matr[i]);
            }
            free(matr);
            system("cls");
            return 0;
        default:
            printf("Ошибка! Не существующий номер операции!Повторите попытку\n");
        }
    }

}

int graph_2() {
    int n = 0;
    printf("Введите количество вершин: ");
    scanf_s("%d", &n);
    int** matr = (int**)malloc(sizeof(int*) * n);
    for (int i = 0; i < n; i++)
        matr[i] = (int*)malloc(sizeof(int) * n);

    int rebra = 0, orientier = 0;
    do {
        printf("\nГраф ориентированный? Введите 1 если да, иначе 0: ");
        scanf("%d", &orientier);
        if (orientier > 1 || orientier < 0)
```

```
                printf("Ошибка! Введите один из предложенных вариантов!\n");
        } while (orientier > 1 || orientier < 0);
        if (orientier == 0) {
                for (int i = 0; i < n; i++)
                {
                        matr[i][i] = 0;
                        for (int j = i + 1; j < n; j++) {
                                do {
                                        printf("Введите расстояние %d - %d (если 0 то между вершинами нет никакого пути): ", i + 1, j + 1);
                                        scanf("%d", &rebra);
                                        if (rebra < 0)
                                                printf("Ошибка! Расстояние должно быть положительным значением!\n");
                                } while (rebra < 0);
                                matr[i][j] = rebra;
                                matr[j][i] = rebra;
                        }
                }
        }
        else {
                for (int i = 0; i < n; i++)
                {
                        matr[i][i] = 0;
                        for (int j = i + 1; j < n; j++) {
                                do {
                                        printf("Введите расстояние %d - %d (если 0 то между вершинами нет никакого пути): ", i + 1, j + 1);
                                        scanf("%d", &rebra);
                                        if (rebra < 0)
                                                printf("Ошибка! Расстояние должно быть положительным значением!\n");
                                } while (rebra < 0);

                                if (rebra != 0) {
                                        do {
                                                printf("\nВ какую сторону провести путь?\nВведите (1) если только из %d - %d, (2) если только из %d - %d, (3) если в обе стороны: \n", i + 1, j + 1, j + 1, i + 1);
                                                scanf("%d", &orientier);
                                                if (orientier > 3 || orientier < 1)
                                                        printf("Ошибка! Введите один из предложенных вариантов!\n");
                                        } while (orientier > 3 || orientier < 1);
                                        if (orientier == 1) {
                                                matr[i][j] = rebra;
                                                matr[j][i] = 0;
                                        }
                                        if (orientier == 2) {
                                                matr[i][j] = 0;
                                                matr[j][i] = rebra;
                                        }
                                        if (orientier == 3) {
```

```
                                        matr[i][j] = rebra;
                                        matr[j][i] = rebra;
                                }
                        }
                        else {
                                matr[i][j] = rebra;
                                matr[j][i] = rebra;
                        }
                }
        }
}


        printf("\n      Матрица:\n№  ");
        for (int i = 0; i < n; i++) {
                printf(" %3d", i + 1);
        }
        printf("\n__|");
        for (int i = 0; i < n; i++) {
                printf("____");
        }
        printf("\n");
        for (int i = 0; i < n; ++i) {
                {
                        printf("%d | ", i + 1);
                        for (int j = 0; j < n; ++j)
                                printf("%3d ", matr[i][j]);
                        printf("\n");
                }
        }
        int opr = 0;
        while (1) {
                printf("\nВведите номер операции:\n1)Выбрать с какой вершины надо найти кратчайшие пути
\n2)Выйти из работы с данным графом\n");
                printf("Выбор: ");
                scanf_s("%d", &opr);
                switch (opr) {
                case 1:
                        dejkctr(matr, n);
                        _getch();
                        break;
                case 2:
                        do{
                        printf("Нужно ли сохранять сгенерированную матрицу? Если да введите 1, если не надо,
то 0: ");
                        scanf("%d", &opr);
                        if (opr > 1 || opr < 0)
                                printf("Ошибка! Введите один из предложенных вариантов!\n");
                } while (opr > 1 || opr < 0);
                        if (opr == 1)
                                save_graph(matr, n);
```

```
                        for (int i = 0; i < n; i++) {
                                free(matr[i]);
                        }
                        free(matr);
                        system("cls");
                        return 0;
                default:
                        printf("Ошибка! Не существующий номер операции!Повторите попытку\n");
                }
        }
}

int graph_3() {
        FILE* F, *S;
        int n = 0;
        int vibor = 0;
        F = fopen("Matrica.txt", "r");
        if (F == NULL) {
                perror("Ошибка при открытии файла");
                while (1) {
                        printf("Хотите создать файл? Введите 1 если да, если нет, то 0:");
                        scanf("%d", &vibor);
                        switch (vibor) {
                        case 1: {
                                        S = fopen("Matrica.txt", "w");
                                        printf("Файл создан, теперь вручную заполните матрицу, сначала
укажите количество вершин, а потом и саму матрицу.");
                                        _getch();
                                        fclose(S);
                                        return 0;
                                }
                        case 0:
                                return 0;
                        default:
                                printf("Ошибка! Не существующий номер операции!Повторите попытку.\n");
                        }
                }
        }

        fscanf(F, "%d", &n);
        if (n == 0) {
                printf("Заполните матрицу! Для этого в  файле 'Matrica.txt' введите сначала количество вершин, а
потом и саму матрицу.");
                _getch();
                return 0;
        }
        int** matr = (int**)malloc(sizeof(int*) * n);
        for (int i = 0; i < n; i++)
                matr[i] = (int*)malloc(sizeof(int) * n);

        for (int i = 0; i < n; i++) {
```

```
		for (int j = 0; j < n; j++)
			matr[i][j] = 0;
	}


	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			fscanf(F, "%d", &matr[i][j]);
			if (matr[i][j] < 0) {
				printf("Ошибка! Данный алгоритм работает только с ребрами положительного веса!\nИзмените значения на положительные!");
				_getch();
				return 0;
			}
		}
	}

	fclose(F);


	printf("\n	Матрица:\n№  ");
	for (int i = 0; i < n; i++) {
		printf(" %3d", i + 1);
	}
	printf("\n__|");
	for (int i = 0; i < n; i++) {
		printf("____");
	}
	printf("\n");
	for (int i = 0; i < n; ++i) {
		{
			printf("%d | ", i + 1);
			for (int j = 0; j < n; ++j)
				printf("%3d ", matr[i][j]);
			printf("\n");
		}
	}
	int opr = 0;
	while (1) {
		printf("\nВведите номер операции:\n1)Выбрать с какой вершины надо найти кратчайшие пути \n2)Выйти из работы с данным графом\n");
		printf("Выбор: ");
		scanf_s("%d", &opr);
		switch (opr) {
		case 1:
			dejkctr(matr, n);
			break;
		case 2:
			for (int i = 0; i < n; i++) {
				free(matr[i]);
			}
```

```
                        free(matr);
                        system("cls");
                        return 0;
                default:
                        printf("Ошибка! Не существующий номер операции!Повторите попытку\n");
                }
        }

}


int main()
{
setlocale(LC_ALL, "Rus");
zast();
int opr = 0;
char* opr1 = {};


while (1) {
        printf("Какой вариант работы с графом выбрать?\n");
        printf("Введите номер операции:\n1)Случайно сгенерировать граф\n2)Создать граф вручную\n3)Загрузить
граф из файла\n4)Завершить работу\n");
        printf("Выбор: ");
        scanf_s("%d", &opr);
        switch (opr) {
        case 1:
                system("cls");
                graph_1();
                break;
        case 2:
                system("cls");
                graph_2();
                break;
        case 3:
                system("cls");
                graph_3();
                break;
        case 4:
                return 0;
        default:
                printf("Ошибка! Не существующий номер операции!Повторите попытку\n");

        }
}
}
```